increasinG tRust with eId for Developing buSiness

D4.2 Guidelines for end-users to instantly connect with "eIDAS enabled, KYC-as-a-service"

Document Identification			
Status	Final	**Due Date**	31/05/2021
Version	1.0	**Submission Date**	30/06/2021

Related Activity	Act 4	**Document Reference**	D4.2
Related Deliverable(s)			
Lead Participant	Kompany	**Lead Author**	Peter Bainbridge-Clayton (kompany)
Contributors	INFOCERT UAegean	**Reviewers**	Nikos Triantafyllou (UAGEAN), Juan Carlos Pérez Baún (Atos)

Keywords
eIDAS, AML, KYC, KYB, GRIDS, Aggregate, BAA

This document is issued within the frame and for the purpose of the GRIDS project. This project has received funding from the European Union's Innovation and Networks Executive Agency – Connecting Europe Facility (CEF) under Grant Agreement No INEA/CEF/ICT/A2019/1926018; Action nº 2019-EU-IA-0044. The opinions expressed and arguments employed herein do not necessarily reflect the official views of the European Commission.

Document Information

List of Contributors	
Name	Partner
Peter Bainbridge-Clayton	Kompany

Document History			
Version	Date	Change editors	Changes
0.1	19/06/2021	Peter Bainbridge-Clayton(kompany)	First draft.
0.2	27/06/2021	Peter Bainbridge-Clayton (kompany)	Updates after review
0.3	29/06/2021	Kompany	Version without comments and changes ready for QA
1.0	30/06/2021	Juan Alonso, Juan Carlos Perez (ATOS)	Review of final version before submission

Quality Control		
Role	Who (Partner short name)	Approval Date
Deliverable leader	Peter Bainbridge-Clayton (Kompany)	29/06/2021
Peer reviewers	Nikos Triantafyllou (UAGEAN) Juan Carlos Pérez Baún (Atos)	29/06/2021
Quality Manager	Juan Alonso (Atos)	30/06/2021

Table of Contents

List of Figures

List of Acronyms

Abbreviation / acronym	Description
AML	Anti Money Laundering
API	Application Programming Interface
CDD	Customer due diligence
DC	Data Consumer
DP	Data Provider
DS	Data Subject
EC	European Commission
eIDAS	eIDAS Regulation (EU) No. 910/2014 on electronic identification and trust services for electronic transactions in the internal market
FIs	Financial Institutions
GDPR	General Data Protection Regulation
JWKS	JSON Web Key Set
KYB	Know Your Business
KYC	Know Your Customer
LEI	Legal Entity Identifier
LOU	Local Unit Operator
OIDC	OpenID Connect
OIDC IDA	IDA OpenID Connect for IDentity Assurance
SDK	Software Development Kit

Executive Summary

GRIDS identifies the importance of bundled services for worldwide commerce, business and financial sectors that require KYC – Know Your Customer guidelines and identifies the service receiver through the trans-European eIDAS eID system.

As a result, part of the project focuses on existing services, whose addressable market is really large and will have a significant growth, especially due to the Covid pandemic period.

Deliverable "D4.2 – Guidelines for end users to instantly connect with eIDAS enabled KYC as a service" is related to Activity 4 of the GRIDS project (increasinG tRust with eId for Developing buSiness).

The goal of this deliverable is to provide information which would enable a Data Consumer product team, coupled with a project management team and a competent development team to integrate into the GRIDS platform, to enhance the offerings they currently provide for their business customers.

Such a consumer might be a bank, seeking to establish the identity and bona fides of a natural person claiming to represent a legal entity – in such cases, there are three aspects to the KYC process:

- Establish and validate the identity of the natural person
- Establish and validate the identity of the legal body
- Establish and validate the relationship of the natural person with the legal body

The goal of GRIDS is to enable all three aspects of this process to be covered as an adjunct to existing eIDAS processes and features – the natural person can be validated through the existing eIDAS functionality, whilst the latter two aspects can be covered through the services provided by DP's and the GRIDS platform.

1 Introduction

1.1 Purpose of the document

This document refers to the Activity 4 – End User APIs for Business Users of GRIDS Services, and more specifically, to task 4.2 dedicated 'Guidelines for end users to instantly connect with eIDAS enabled KYC as a service'.

The GRIDS platform enables connectivity between the eIDAS platform, Data Consumers and Data Providers. It extends the functionality provided by the eIDAS platform to include information sourced from Data Providers to legal persons. In the GRIDS architecture, an 'End User' is referred to as a DC – a Data Consumer. Data Consumers use the BAA (Business Attribute Aggregator) component of the GRIDS platform to authenticate natural persons through the eIDAS system, and to authenticate legal persons and their relationship with natural persons through a network of Data Providers. GRIDS specifies the interfaces between parties to ensure compatibility between different DCs and DPs.

1.2 Relation to other project work

All deliverables related to this project are depended on:

- Activity 2:
  - Task 2.2: Architecture and APIs Design and action plan
  - Deliverable 4.3: APIs Development and Deployment
- Activity 3:
  - Task 3.1: BAA Development
- Activity 4:
  - D4.3 APIs Development and Deployment

1.3 Structure of the document

This document is structured in 4 major chapters:

Chapter 2 presents an overview of the manner of connection into GRIDS

Chapter 3 presents more detail on the use of the GRIDS SDK

Chapter 4 presents the conclusions of the deliverable.

2 GRIDS Connection

The Data Consumers can use either the published interfaces, based on OpenID Connect for Identity Assurance, or to simplify the connection, they can make use of the GRIDS prebuilt SDK.

Data Consumer created connections based on the published specification are out of scope of this document, but some prerequisites are the same, as detailed below in **¡Error! No se encuentra el origen de la referencia.**. Also, reading this document will help a non-SDK user to understand the flows required to interface to the GRIDS platform.

Data Provider connections into the GRIDS platform are out of scope for this document.

2.1 Data Consumer prerequisites

Any party wishing to connect into the GRIDS platform should be familiar with the concepts of encryption, digital signing, interfaces, and the operation of the existing eIDAS system upon which GRIDS is based. Familiarity with OIDC (OpenIdConnect for Identity Assurance) and OAuth 2.0 would be a benefit.

Data Consumers wishing to use the provided GRIDS SDK should be familiar with Java development and use of JARs within a project.

2.2 Data Consumer prerequisites

GRIS SDK uses JAVA as its main language and a JAR package file is generated and shared with the Data Consumers.

The main purpose of the SDK is providing a set of interfaces, models and services(wrappers) following OpenID Connect for Identity Assurance and eKYC specifications that the DC can use in order to connect and consume GRIDS API endpoints.

The SDK provides two main classes for the connection management, **GRIDSClientManager** which handles the client level interactions, and **GRIDSIssuer** which handles the transaction level interactions. The SDK also makes use of OIDC defined classes including **OIDCClientInformation , OIDCClaimsRequest , OIDCTokens , OIDCProviderMetadata, ClientInformation , UserInfo**

The SDK classes are described in more detail below, but the general flow is as follows:

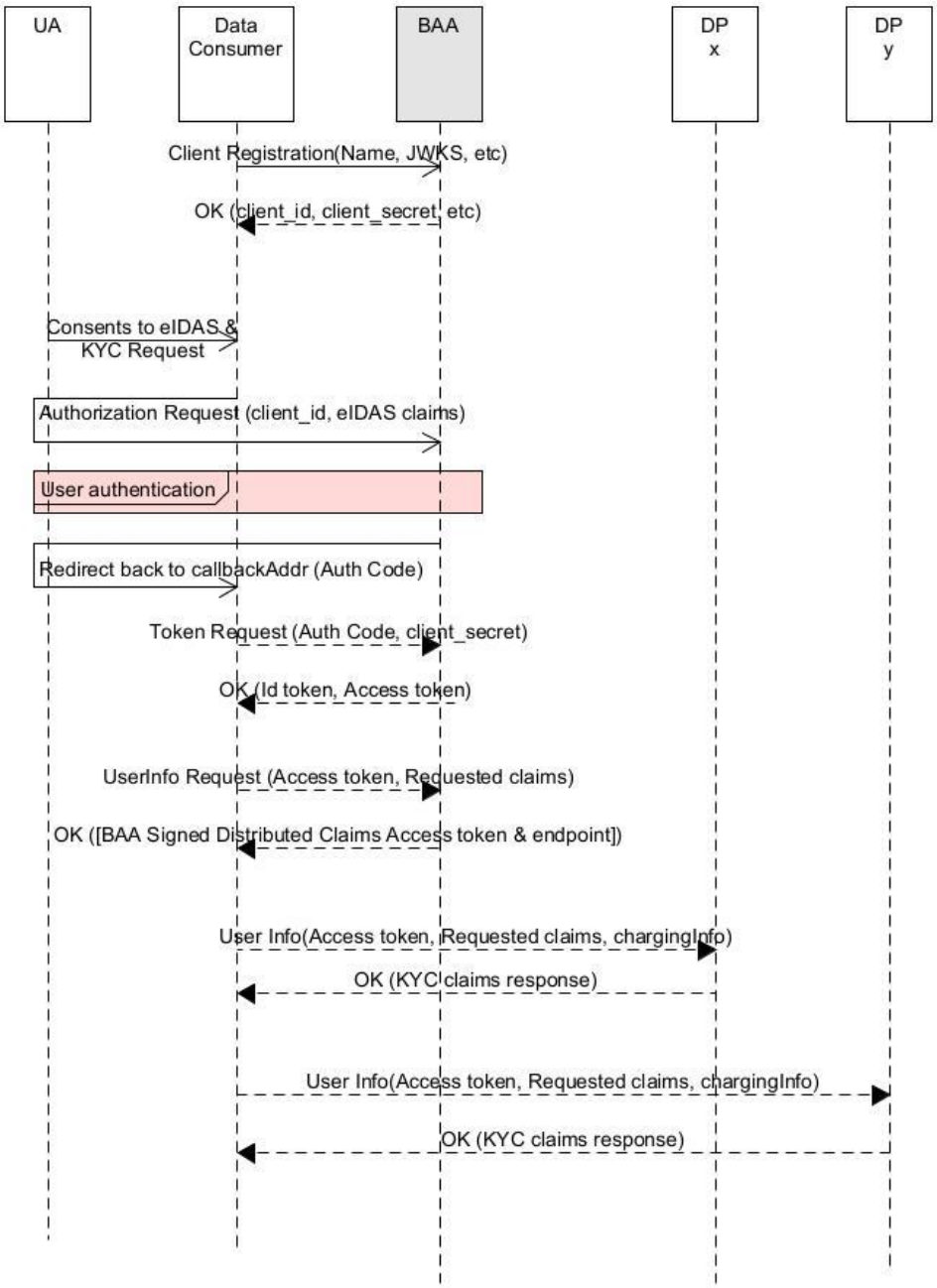

Figure 1: SDK Flow Diagram

1. Use the GRIDSClientManager to perform the OpenID Connect Client Registration, and obtain a client_id and a secret
2. Obtain consent from the user to use eIDAS and GRIDS for authentication
3. Initialise a GRIDSIssuer object
4. Use the GRIDSIssuer object to get an OIDCProviderMetadata object
5. Use the **getAuthorizationUrl** method of the **OIDCProviderObject** to redirect the end users browser for them to complete registration or login
6. The Data Consumers callback URLs will be triggered from GRIDS passing in accesses tokens

7. The tokens can be used in the **GRIDSIssuer getUserInfo** method which will return a **UserInfo** object
8. The **UserInfo** object contains information about the distributed claims, which can be used in the **GRIDSIssuer** object **getDPUserInfo** method for appropriate Data Providers.
9. This will return a new UserInfo object representing the claims validated by the appropriate data provider

2.3 Data Handling and Commercial Relationships

It is important to note that the response from different data providers is not proscribed by the GRIDS specification, and also that a commercial relationship may be required by DPs in order to provide service. It is possible therefore that a DP may not be presented in the distributed claims UserInfo response, but also that it may be, but the actual response from the DP is not available.

The first UserInfo response should therefore be taken as a list of DP's which may be able to satisfy the request, not a list of DPs that have agreed to handle the request.

In instances where a commercial relationship is required between a DC and a DP, the nature and establishment of that relationship is beyond the scope of this document and of the GRIDS project in general.

Given the different data sets and proof types available from different Data Providers, the actual content is defined in a limited way, but does not preclude any data provider from extending the response with other data points or structures. This will be DP dependent, and could be DC dependent in the event of a commercial relationship between the parties. Such extensions are therefore outside the scope of this document.

3 SDK Detail

As described above, the SDK uses predefined OIDC calls, and extends them with GRIDS specific classes which abstract the detail of the GRIDS system to the developer. These classes and their usage are described in more detail below.

3.1 GRIDSClientManager

Client applications must be registered with OpenID Connect Dynamic Client Registration, which extends OAuth 2.0 Dynamic Client Registration Protocol (https://datatracker.ietf.org/doc/html/rfc7591) and OAuth 2.0 Dynamic Client Registration Management Protocol (https://datatracker.ietf.org/doc/html/rfc7592) before they can send authorisation requests to it. GRIDSClientManager wraps this logic and helps Data Consumer to dynamically register itself with the Data Consumer Connector.

3.1.1 Constructors

GRIDS Client Manager (URI clients Endpoint) - Initializes a new instance of the GRIDSClientManager

3.1.1.1 Parameters

Parameter	Type	Description	Required
clientsEndpoint	URI	OpenID provider client registration endpoint	true

3.1.2 Properties

Property	Type	Description
clientsEndpoint	URI	OpenID provider client registration endpoint

3.1.3 Methods

Method	Type	Description
registerClient (clientMetadata, masterToken)	OIDCClientInformation	Register's client

3.2 GRIDSIssuer

GRIDSIssuer encapsulates a discovered OpenID Connect Provider and its metadata. Provides the methods for getting an authorization URL, consuming callbacks, triggering the token endpoint and getting userInfo from BAA and DP.

3.2.1 Constructors

GRIDSIssuer (URI, String, String, URI) - Initializes a new instance of the GRIDSIssuer

3.2.1.1 Parameters

Parameter	Type	Description	Required
endpointURI	URI	OpenID provider metadata URL	true

clientId	String	The client identifier issued to the client during the registration process	true
clientSecret	String	The client secret	true
callbackURI	URI	Client's redirection endpoint previously established with the authorization server during the client registration process	true

3.2.2 Properties

Property	Type	Description
endpointURI	URI	OpenID provider metadata URL
clientId	String	The client identifier issued to the client during the registration process
clientSecret	String	The client secret
callbackURI	URI	Client's redirection endpoint previously established with the authorization server during the client registration process

3.2.3 Methods

Method	Type	Description
getOPMetadata()	OIDCProviderMetadata	Requests OPMetadata from DCC
getAuthorizationUrl(OIDCClaimsRequest claims)	String	Returns the complete URI representation for DC to redirect in order to start the authorization
requestToken(String callbackResponse)	OIDCTokens	Requests Token from DCC. Code etc will be parsed from url. Url ex. https://dc.com/callback?state=12345G8&code=aaaabbbbbbccccccddd
getUserInfo(token)	UserInfo	Requests userinfo from DCC
getDPUserInfo(userInfo URI,token)	UserInfo	Requests userinfo from DP

A description and reference of each type used can be found on chapter 3.4.

3.3 SDK call flow

1) Data Consumer uses **GRIDSClientManager** to register a new client with DCC and receives a client id and a client_secret. That can be achieved by calling **registerClient** and passing all required parameters, example: name, DC JWKS enpoint, callback url, etc. A Client Registration example can be found below:

```
URI clientURL = new URI("https://dcc.grids.eu/clients ");
GRIDSClientManager gridsClientManager = new GRIDSClientManager(clientURL);

ClientMetadata clientMetadata = new ClientMetadata();
clientMetadata.setRedirectionURI(URI.create("https://example.com/cb"));
clientMetadata.setJWKSetURI(URI.create("https://example.com/jwks"));
clientMetadata.setName("My Client App");

OIDCClientInformation clientInfo =
gridsClientManager.registerClient(clientMetadata, "master access token");

String clientID = clientInfo.getID().getValue();
```

Figure 2: Client registration example

2) User Consents and wants to login or register with GRIDS.
3) Data Consumer Initializes a GRIDSIssuer using the GRIDS enpointUrl, client_id, client_secret and callbackUrl. After initialization DC can request BAA configuration using getOPMetadata method in order to retrieve all required metadata in order to continue.
   A GRIDS Issuer and OP Metadata example can be found below:

```
URI wellKnownURI = new URI("https://dcc.grids.eu/.well-known/openid-configuration
");
URI callbackURI = new URI("https://dc.example.com/callback");

GRIDSIssuer gridsIssuer = new GRIDSIssuer(wellKnownURI, "client_id",
"client_secret", callbackURI);
```

Figure 3: GRIDS Issuer example

```
OIDCProviderMetadata opMetadata = gridsIssuer. getOPMetadata();

//Read metadata example
URI issuer = metadata.getIssuer(); // ex "https://DCC.example.com"
URI authorizationURI=op.getAuthorizationEndpointURI()//
https://BAA.example.com/connect/authorize
URI tokenURI = op.getTokenEndpointURI() // https://DCC.example.com/connect/token
List<String> st =  metadata.getSubjectTypes() // ["public", "pairwise"]
URI jwks = metadata.getJWKSetURI() // https://BAA.example.com/jwks.json
Boolean svc = metadata.supportsVerifiedClaims() // true
List<String> itf = metadata.getIdentityTrustFrameworks() //
["eidas_ial_substantial", "eidas_ial_high"]
List<String> idt = metadata.getIdentityDocumentTypes() // ["idcard", "passport"]
List<String> ivm = metadata.getIdentityVerificationMethods() // ["pipp", "sripp",
"eid"]
List<String> vcs = metadata.getVerifiedClaims() // ["given_name", "family_name",
```

Figure 4: OP Metadata example

4) DC will call **getAuthorizationUrl** passing all the request claims as a parameter. Doing so, a url will be returned. Using this url DC will redirect Users browser in order for the user to complete register or login.

A Get Authorization Url example can be found below:

```
JSONObject verification = new JSONObject();
verification.put("trust_framework", null);

OIDCClaimsRequest claimsRequest = new OIDCClaimsRequest()
    .withUserInfoVerifiedClaimsRequest(
    new VerifiedClaimsSetRequest()
    .withVerificationJSONObject(verification)
    .add("given_name")
    .add("family_name")
    .add("address")

URI request = gridsIssuer. getAuthorizationUrl (claimsRequest);

//redirect users browser
```

Figure 5: Get Authorization Url example

5) After a user successfully logins or registers, DCs callbackUrl will be triggered. Passing that into **GRIDSIssuer requestToken** will request from DCC and return an Access token.

A Request Token example can be found below:

```
// The request url from OIDC Provider to Data Consumer
// https://dc.com/callback?state=12345G8&code=aaaabbbbbbccccccddd
OIDCTokens tokens = gridsIssuer. requestToken (requestUrl);

JWT idToken = tokens.getIDToken();
AccessToken accessToken = tokens .getAccessToken();
RefreshToken refreshToken = tokens.getRefreshToken();
```

Figure 6: Request Token example

6) Passing that Access Token into **getUserInfo** will request from DCC and return users info including a set of distributed claims.

A Get User Info example can be found below:

```
UserInfo userInfo = gridsIssuer.getUserInfo(token);
String subject = userInfo.getSubject().getValue();

Set<DistributedClaims> set = userInfo.getDistributedClaims();
for (DistributedClaims c : set) {
    String claimName = c.getNames();
    URI dpEndpoing = c.getSourceEndpoint();
    String token = c.getAccessToken().getValue();
}
```

Figure 7: Get User Info example

7) Using those claims, DC can user claim url and claim access token to call **getDPUserInfo**. Dog so GRIDSIssuer will make an API call to the specified DP in order to receive the verified claims.

A Get DP User Info example can be found below:

```
List<String> claimNamesToGet = Arrays.asList("given_name", "family_name",
"birthdate");
Set<DistributedClaims> set = userInfo.getDistributedClaims();
for (String claimName : claimNamesToGet) {
    for (DistributedClaims c : set) {
        Set<String> claimNames = c.getNames();

        if (claimNames.contains(claimName)) {
            URI dpEndpoing = c.getSourceEndpoint();
            String token = c.getAccessToken().getValue();
            UserInfo userInfo = gridsIssuer.getDPUserInfo(token);

            VerifiedClaimsSet verifiedClaims = userInfo.getVerifiedClaims().get(0);
            PersonClaims verifiedPersonClaims = verifiedClaims.getClaimsSet();
            String givenName = verifiedPersonClaims.getGivenName(); // Max
            String familynName = verifiedPersonClaims.getFamilyName(); // Meier
            String birthDay = verifiedPersonClaims.getBirthdate(); // 1956-01-28
        }
    }
}
```

Figure 8: Get DP User Info example

4 Conclusions

The purpose of this document is to provide the main points used by a Data Consumer to make use of the facilities offered by GRIDS. It is not intended that this document can cover every method of interfacing to the GRIDs system, or to cover every possible programming language that may be used to connect to the platform.

This document offers information suitable for a competent development team to connect into and use the GRIDS platform, whether they choose to use the prebuilt JAVA SDK or to roll their own interface, but it is recommended that the SDK is used, if possible. In this manner the DC can benefit from bug fixes, enhancements etc.